Audio Programming 2

Johan Pauwels

johan.pauwels@uwl.ac.uk

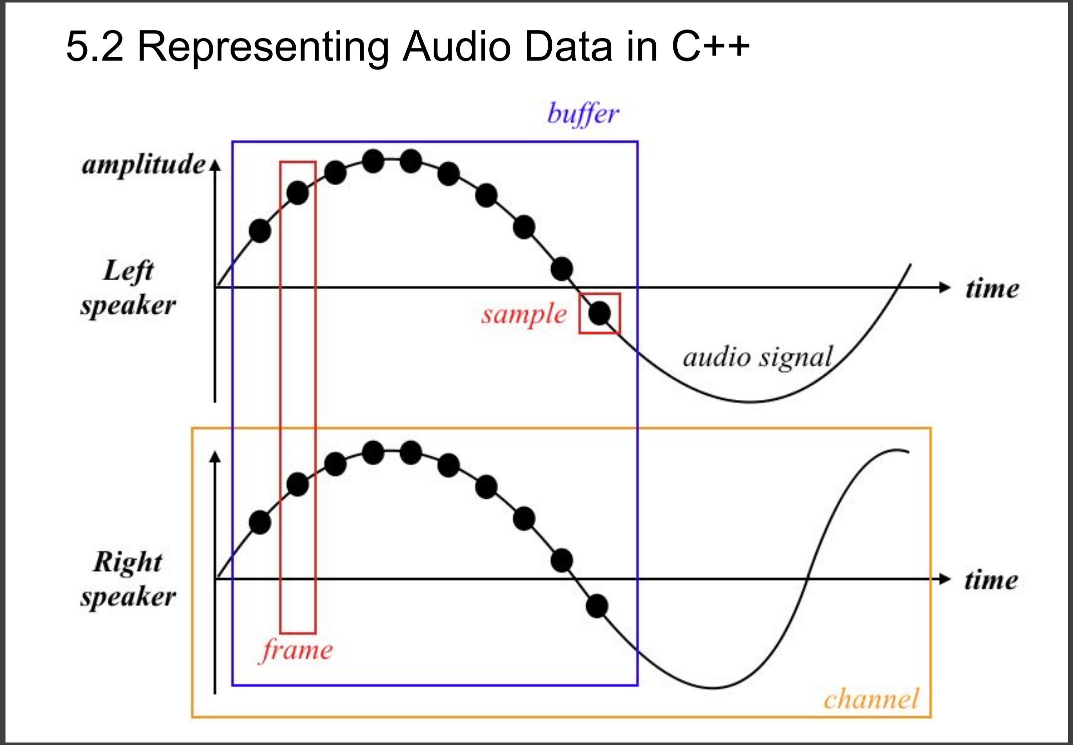
On Today’s Programme

Datatypes and samples

Audio buffers

Representing digital audio

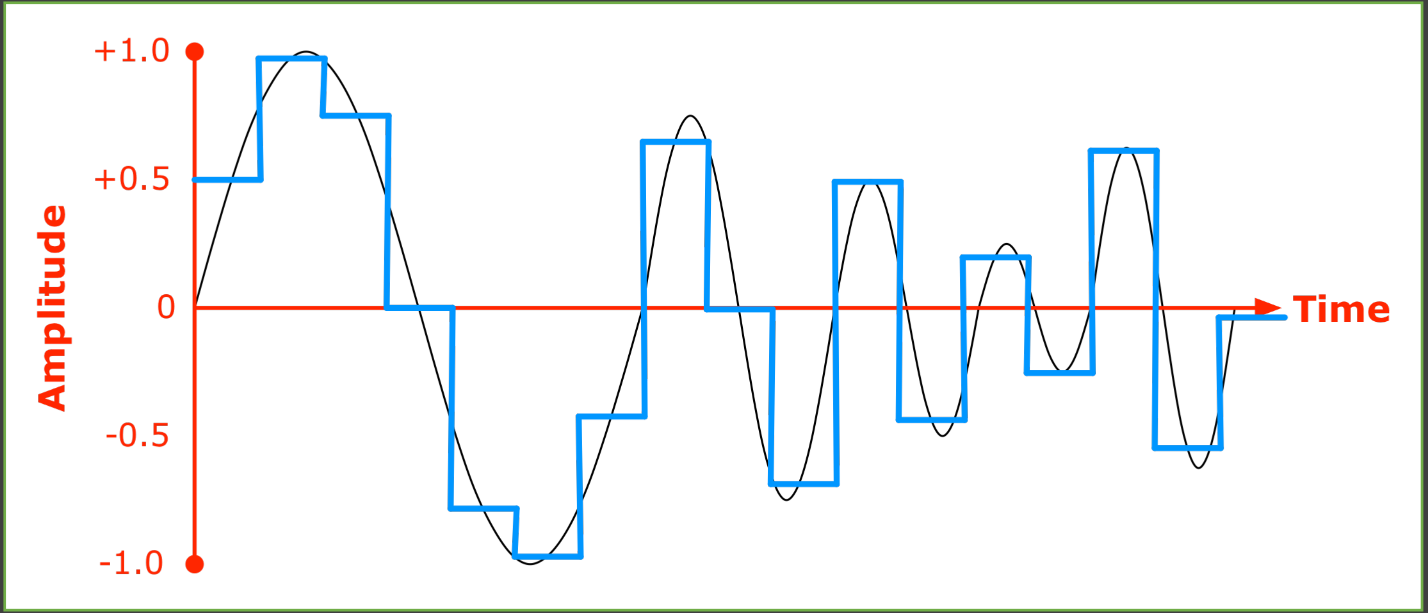
- Extract from “A Standard Audio API for C++: Motivation, Scope, and Basic Design”



5.2 Representing Audio Data in C++

Representing digital audio

- More reference material: “Digital audio concepts” on Mozilla Developer Network



(first two sections, up to “Audio compression basics” although rest is interesting too

Samples in audio processing

- audio data is normally represented with a given data type across an application
- Sometimes this data type is typedefined in some header file, e.g.:

  ```
  typedef double sample_t;
  ```

  Sometimes this type definition can even be done conditionally depending on some compile time settings, e.g.:

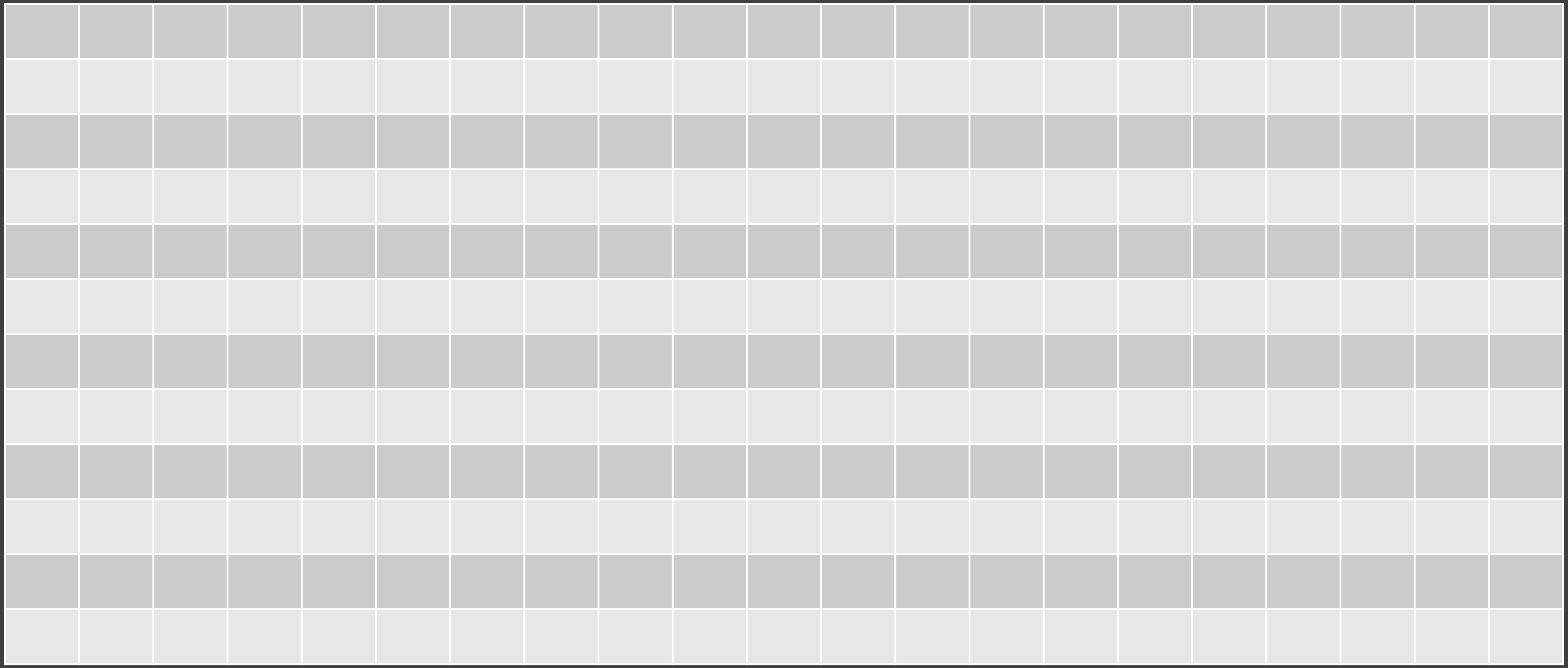
  ```
  #ifdef USE_DOUBLE_PRECISION
  typedef double sample_t;
  #else
  typedef float sample_t;
  #endif
  ```

- Sometimes, it is just #define'd, e.g.:

  ```
  #define sample_t float
  ```

- When programming for that specific application, you should use the publicly available definition and not directly the underlying data type, in order to ensure portability.
- Common types for audio samples are:
  - float, double on CPUs and higher-end microcontrollers
  - int32_t, int16_t on lower-end microcontrollers, or some DSPs

Managing memory



Buffers in audio processing

- when audio data is passed around within an application, this is normally done in buffers (i.e.: blocks of data containing multiple audio samples from one or more channels)

- almost never will you copy audio data between functions: buffers can be large, and copying takes time
  - Pass around the location where to find data, not data itself

- buffers can be passed around without copying as references or raw pointers

Why buffer audio?

- Trade-off between latency and stability
  - Outputting samples as soon as they are ready would avoid buffer-induced latency (other sources of latency still exist)
  - No guarantees that samples will be ready in time (especially with general purpose OS)
  - Some algorithms can’t work on samples in isolated but need local context (and some even complete tracks, making real-time operation impossible)

Buffer datatypes

- float buffer[123], or
  float* buffer = new float[123];
  a pointer to a location in memory that contains some data of type float
- std::array<float, len> buffer : basically a wrapper around a C-style array, but the size has to be known at compile time (and some other C++ benefits)
- std::vector<float> buffer:
  manages its own memory, can resize on demand, has [] operator overload, you can directly access the underlying raw float* pointer
- These are all wrappers around some memory

Multi-dimensional buffers

- float**: a pointer to a pointer to a float
- std::vector<std::vector<float>>: a vector of vectors of floats

Buffer

- A buffer containing samples from a single channel will normally look like:
  float buffer[4] = {A0, A1, A2, A3};
  float* ptr = buffer

- Where:
  - A is the channel (and there is only one for now)
  - 0,1,2 ... correspond to the *frame*, that is the number of sampling periods that intercur between that sample and the first one
- Quiz:
  - What is the address of A3?
  - With a sampling rate of 48kHz, if A0 was sampled at t = 0s, when was A3 sampled?

Answers

- What is the address of A3?
  - A buffer containing samples from a single channel will normally look like:

```
float buffer[4] = {A0, A1, A2, A3};
float* ptr = buffer
ptr == &buffer[0];
ptr + 1 == &buffer[1];
ptr + 2 == &buffer[2];
ptr + 3 == &buffer[3];
```

- Where:
  - A is the channel (and there is only one for now)
  - 0,1,2 ... correspond to the *frame*, that is the number of sampling periods that occur between that sample and the first one

Answers

float buffer[4] = {A0, A1, A2, A3};

- Where:
  - A is the channel (and there is only one for now)
  - 0,1,2 ... correspond to the *frame*, that is the number of sampling periods that occur between that sample and the first one

- With a sampling rate F_S = 48kHz, if A0 was sampled at t = 0s, when was A3 sampled?
  - T = 1/Fs
  - $t_{A0} = 0$
  - $t_{A1} = t_{A0} + T$
  - $t_{A2} = t_{A0} + 2T$
  - $t_{A3} = t_{A0} + 3T = 62.5us$

Exercise

- Write a C++ class “FloatVector” that works as a pseudo re-implementation of std::vector (for float only). This shall include at least:
  - constructor
  - destructor
  - push_back()
  - operator[]()
  - size()
  - capacity()
  - resize()
  - reserve()
- Learn how to use reference documentation:
  - https://en.cppreference.com
  - https://www.cplusplus.com/reference/